Coupling Application design and User interface design

Dennis J.M.J. de Baar & James D. Foley

College of Computing
Georgia Institute of Technology
foley@cc.gatech.edu

Kevin Mullet

Human Interface Technology Group
Sun Microsystems Inc.
mullet@eng.sun.com

june 1991

Abstract

Building an interactive application requires, amongst other activities, the design of both a data model and a user interface. These two designs are often done separately, frequently by different individuals or groups. However, there are strong similarities between the objects, actions and attributes of the data model and those of the user interface. This means that considerable specification work has to be done twice. Our approach is to automatically generate user interface elements directly from the data model. This saves time and effort and at the same time style rules can be applied automatically to the design of the user interface. This relieves the designer of the need to be familiar with certain style rules, while at the same time creating the consistency embodied in the rules.

Introduction

An early step in the design of an interactive application is the definition of the application's data model. Assuming an object oriented design, the data model consists of an object class hierarchy, in which objects have attributes and methods. Single or multiple inheritance is typically used to avoid repetitive specification of shared methods and attributes. Attributes and methods of objects are either internal or external. Internal attributes and methods are not shown in the user interface, but are merely meant for use within the application. External attributes and methods are represented in the user interface and as well as being used within the application. These external attributes and actions are either shown as widgets (i.e. sliders, choices, etc.) or by means of user defined representations.

A later step is the design of the user interface. An increasing number of software tools assist the designer in this step. They provide standard user interface elements and allow the designer to interactively lay out user interface elements on the screen. Examples of user interface builders are DevGuide [Sun90a], the Interface Architect [Hewl90]. The use of these tools, however, does not guarantee good user interface design; that depends on the designer. Guidelines for presenting an application in a tasteful and functional manner are embodied in a number of user interface styleguides, such as the OPEN LOOK Graphical User Interface Application Style Guidelines [Sun90b], the OSF/Motif Style Guide [OSF], the Apple Desktop Interface [Appl86] and IBM's styleguide [IBM87]. These styleguides help the developer design a user interface based on principles of simplicity, consistency and efficiency.

With each of these tools, however, the user interface designer has to perform three unnecessary steps. First, he must access, either from documentation or from his memory, details of the data model. Second, he must access and apply the style guide rules which determine how each element of the data model is to be mapped into a control widget. Finally, he must access and apply layout rules concerning the placement of each control widget.

For example, to lay out a property window for an object, he must know the attributes (properties) of the object, decide which widget to use for each attribute, and lay out the widgets in a window.

Automatic generation of dialog boxes from a high-level textual specification description provides the possibility to automatically apply layout style rules to the design, avoiding some of the unnecessary steps described above. Jade [Zand90], ITS [Wiec90] and Mickey[Olse89] all generate dialog boxes from textual specifications. Mickey generates Macintosh user interface (menu's and dialog boxes) from interface descriptions embedded in Pascal. ITS and Jade use a set of style rules, created by a style expert and a specification of the dialog content to generate dialog boxes. Jade also includes a graphical editing capabilities, which allow the designer to refine the generated user interface.

Our system automates all the three steps, using D2M2edit[-Beek90], a tool for creating data models; DevGuide [Sun90a], Sun Microsystems' interactive user interface lay-

out tool, and the OPEN LOOK Style Guide Rules [SUN90b].
We will first describe our mechanism used to transform selected application data as dialog content into user interface dialog. After that we expain the relation between data in the application model and the dialog parts of the user interface. Finally, an overview between the two design stages will be described.

Transforming Application data to Interaction Objects

Descriptions of data from the application data model, actions and attributes,are used as input; a set of interaction conclusions is generated as output (Figure 2). This intelligent design link focuses on automatic organization and development of menus and dialogue boxes using various types of knowledge and a set of design rules as engine.

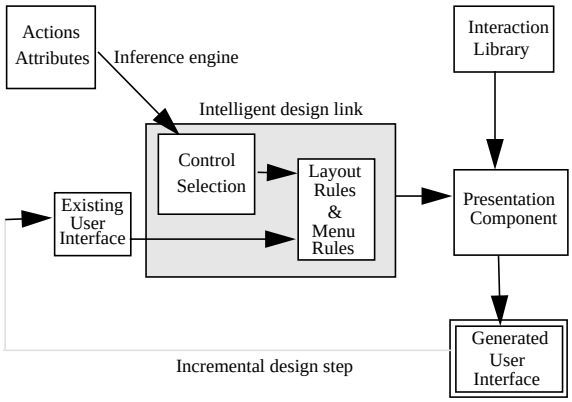


Figure 1 Information flow in automatic generation process

The rules for control selection are implemented as a linear list of simple ‘If Then’ rules. The inference engine is implemented so that it can be used independent of the interaction library. The engine produces general conclusions which have to be transformed with an appropriate presentation component for the designated library. For every action a menu advise is generated.
To support incremental design already existing user interface can be used as input for the design tool . For menu configuration the design tool uses the existing user interface to determine where the different actions have to be placed in the different existing menus.

In the next section I describe how application attributes and actions relate to dialog boxes and interaction objects.

Application objects & Interaction Objects

In the data model of the application, every object has attributes and actions on the object. The actions are associated with pre- and postconditions. The data model is based on that of UIDE[Fole88].

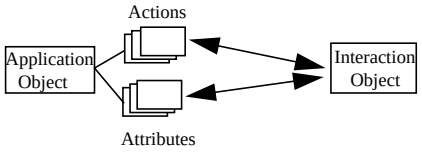


Figure 2 The relation between application data and interaction objects

Every user interface interaction object is related to an action or attribute in the application data model (Figure 1). Interactions objects are pre-defined user interface objects (button, slider, menu, etc.).

Attribute: ***integer***

range[0..10]
precision[low]
length[2]
name[VolumeInput]
label[Volume]

Example 1 An attribute from the application model and its representation of an interaction object in the user interface

In the example above (Example 1) the attribute ‘VolumeInput’ has a graphical slider to change its value. We added an additional descriptor ‘precision’ to the attributes data. Without this parameter the choice between the different available widgets for ‘VolumeInput’ would have been based on the preference setting in the rules. In this case the engine could have inferred a slider or a numerical text field. The more additional data we add to the application data the more precise a matching widget can be inferred from the rules. We don’t want the application designer have to specify too much additional data. Especially user interface related specifics should not be part of the application data model. In example 1 we needed some semantics of the integer ‘VolumeInput’ for the control selection. These semantics are independent of the look-and-feel of the interaction object. The inference engine chooses the best widget for the data element (as defined in the rules). When more possible widgets can be used/inferred for the data element, then the control selection will pass this information to the presentation component. The presentation component on his turn can use this information to allow regeneration of user interface parts with multiple possibilities. One possible way to do this would be a cycle button for every user interface element with multiple possibilities for the designer to switch between the possible widgets.

The order of the items in a dialog box or menu is currently based on the order, which the data elements are selected in the data model.
We distinguish two different dialog areas. The base window and the popup window. The base window is the main window of the application. All the side information and interaction of an application should go in a popup window. For the setting of properties we use a property window and for the initiation of an action we use a command window. In the base window only menus or single command buttons should be used.

Control selection rules

The control selection is defined as a set of rules, which are derived from the OPEN LOOK style guide book.We use the following user interface controls (Figure 3):

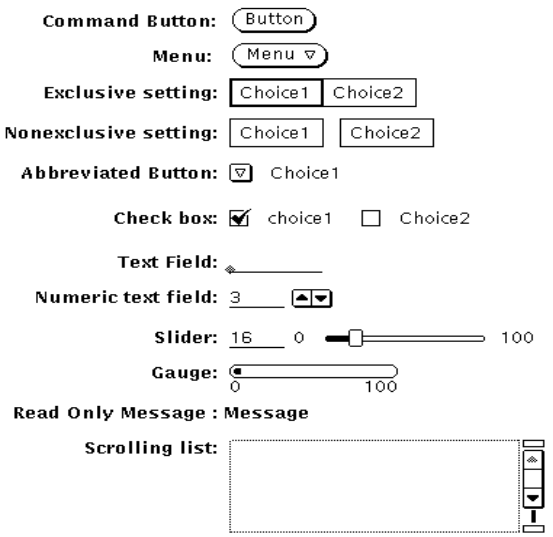


Figure 3 The different control widgets.

In the future it might be useful to extend the list of widgets. In this case we can extend the control selection list with more selection rules. The new widgets also have to be added to the presentation component.
There are now five different attribute types and an action included in the rules for the control selection. This basic set of types was used to test our rule base. In the tables we show some of the control selection heuristics. The tables are not complete, but give an idea of the rules in the rule base.

Label	Widget
	Excl. setting
On/Off	Check box
Yes/No	Check box
True/False	Check box

Table 1 Control Selection for **boolean** attributes; the only descriptor is the label.

The boolean (Table 1) can have only two values. A boolean is always True or False. The labels are always opposite labels. For the boolean attributes we prefer to use a checkbox, because of the look and feel semantics captured in the widget. People are used to mark a check boxes on forms. We have the exclusive setting also reserved as possibility in the rules.
The second attribute is the integer (Table 2). The integer can have so many different purposes, and therefore the control selection is not always precise. In example 1 we used the attribute ‘VolumeInput’ for which the slider is a good interaction widget. For an integer ‘SocialSecurityNumber’ we certainly don’t want to use a slider, but a numerical text input field. In most cases the numerical text field will be the default choice for the integer.

R/W	Precision	Range	Length	Widget
writable	low	unknown	small	Text field
writable	low	known	small	Numeric Text field
writable	low	large	large	Graphical Slider
writable	high	large	large	Slider w/ boxes
read only	low			Gauge
read only	high			Numertc text field

Table 2 Control selection for **integer** attributes; R/W is the writability, a low precision integer doesn’t require a precise input, high precision visa versa, range is the range of the integer, length is the number of digits.

In case of a real value (Table 3) the interaction object will most likely be a normal text input field. In some cases the real variable might require an slider as an interaction object, especially if the number of digits is very high. For the display of values, which are only readable we use a message in the case of a text attribute or a gauge in the case of a numerical attribute.

R/W	Precision	Range	Length	Widget
writable	low	known	high	Slider
writable	low	known	low	Text Field
writable	high			Text Field
read only	low			Gauge
read only	high			Text field

Table 3 Control selection for **real** attributes; the descriptors are the same as the descriptors for the integer (Table 2).

An enumerated list of values (Table 4) can be displayed in the user interface in different ways. The control selection for an enumerated type is based on several properties. We distinguish between mutually exclusive and non-exclusive enumerations. A mutually exclusive enumeration always requires one item to be selected. A non-exclusive enumeration allows multiple items to be selected from a group of items. A variation on a mutually exclusive and a non-exclusive enumeration allows one choice or no choice. Length is an import factor for the control selection. The total length of an enumeration is dependent on it exclusiveness and on the

length of the items. Depending on this length and the number of items a choice is made between the different widgets. For a group of items larger than 9 a scrolling list will be chosen. If the number of items is smaller than 9 then still a scrolling list can be chosen if the length of the items is to large to fit in a popup window and the enumeration is non exclusive. An enumeration is a static list of values. An dynamic list, which can be extended during run-time, requires an dynamic scrollable list. The length in table 4 is in characters, we actually work with proportional fonts.

Values	min	max	Length	Widget
2-9	0	1	21-50	Variation Excl Set
2-9	0	1	50+	Var. Scrolling List
2-9	1	1	21-50	Exclusive Setting
2-9	1	1	50+	Abbreviated Excl. Setting Menu
2-9	0	1+	21-50	Non excl. Setting
2-9	0	1+	50+	Scrolling List
10-18	0	1		Scrolling List
10-18	1	1		Scrolling List
10-18	1	1+		Scrolling List

Table 4 Control selection for **enumerated** attributes; values is the number of values in the enumeration, min and max represent the number of choices reuired, length is the total character length of the enumeration.

The text attribute (Table 5) is translated in a normal text field widget. Based on the length of the text field a single or multi-line text field widget is chosen.

R/W	Length	Widget
writable	0-80	single line text field
writable	80+	multiple line text field
read only		Message

Table 5 Control selection for **text** attributes; R/W is the writability

Buttons are used to issue direct commands. The command window is typically a place to use command buttons. In the property sheet we use the button ‘Apply’, ‘Reset’ and ‘Cancel’ conform the OPEN LOOK standard.

Figure 4 is an example of a property window generated form application data selected form the data model. The control selection configured a widget for every attribute. The attributes ‘Author’ and ‘Book’ are normal text attributes. The first character of each label is a capital and the labels are provided with a colon. The attribute ‘Cover’ is a mutually exclusive enumeration, which means that a book in this case can have only a ‘Hard’ or a ‘Soft’ cover. For the attribute ‘Book id’ numerical arrows were provided. The attribute ‘Setting’ is also a mutually exclusive enumeration of different book types. The buttons ‘Apply’ and ‘Reset’ are standard buttons for all property windows. All these controls are arranged by a set of layout rules. In the popup window (command window, property window) we always use a one column layout. The labels are always on the left side of the control. Every label has a colon and the all the labels are aligned by their colons. We use a grid for the layout. With a 12-point font we use a 10-point grid. Some of the layout rules are:

- 2 grid units of white space on the left edge of the pane and the left of the longest label.
- 2 grid units of white space between the right of the longest control and the right edge of the pane.
- 1 grid unit between the colon and the left of the control.
- 1 grid unit between controls.
- buttons centered at the bottom of the pane, 1 grid unit between them.

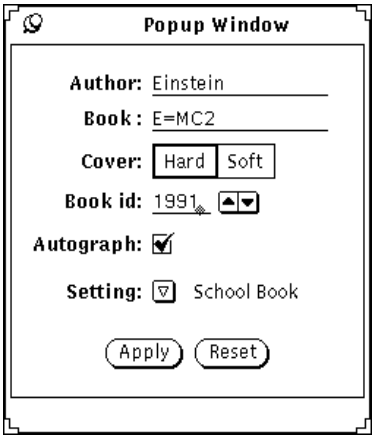
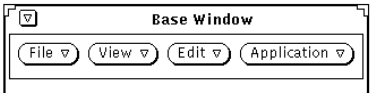


Figure 4 A property window generated by our system. The property window is designed conform the Open Look standard.

For the base window, the main window of an application, there are different layout rules. In the base window we only allow actions, which can either be put in one of the menus or in a single button. Between the different menus and buttons in the menu bar should be one grid unit of white space and the menus should be starting from left to right. In the menu bar we use different standard menus. The order of the standard menus is also based on a style rule. In Figure 4 are the standard menus ‘File‘, ‘View’ and ‘Edit’ and an application-specific menu.
One of the keywords for the design of a user interface is ‘Consistency’. We provide consistency through the applications by using the same menus and the same order of these menus in every application.The user will become familiar with this framework and has to learn only the application specifics.



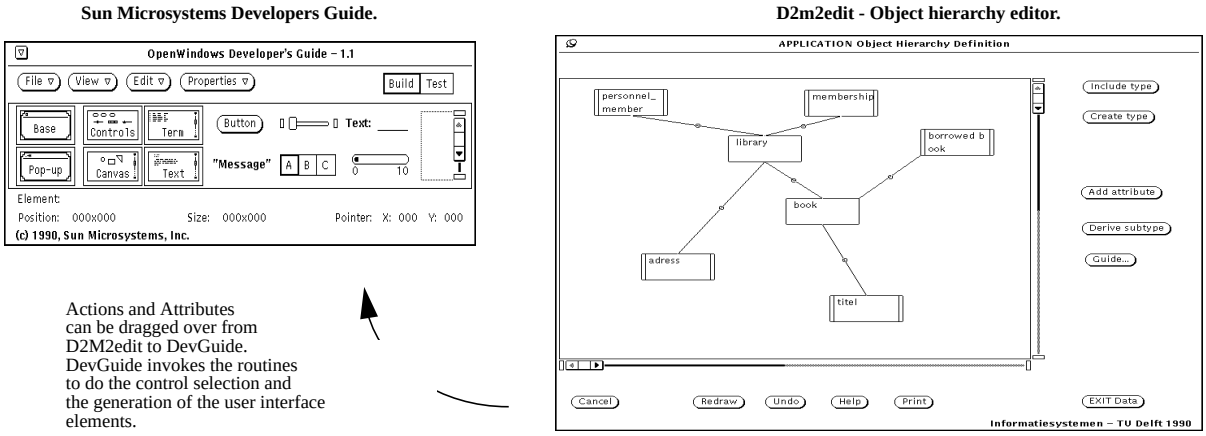


Figure 6 D2m2edit and DevGuide

Figure 5 The standard menu buttons in a pre-defined order. For application specific actions a application menu is generated.

Every action gets a menu advise during the control selection. In the next section the standard menus and menu rules are described.

Standard Menus and Menu Rules

The Open Look standards configure the actions in the application in different standard menus. A standard application has one base window with a menu bar. Depending on the kind of action the action will be placed in one of the following menus.

Filing: Creating, saving, loading and printing files

Anything that affects the entire application (and that could implicitly affect objects that live in the application) is a File function. For example, creating, saving, printing, importing to, and exporting from a database query are all File functions.

Viewing: Controlling how data is displayed in the application

Anything that affects the perspective and the details of the application and the application objects is a View function. Anything that changes the display, size or form is a view function.

Editing: Making changes to data

Anything that affects the existence or state of objects that users can select is an Edit function. For example, deleting, copying, and inserting (existing or previously existing) database elements, such as tables, are Edit functions.

Properties: Setting properties for data, usually for selected anything that changes an object's attributes is a property function. For example, altering the attribute of the database element, such as tables, is a Property function. Such attributes include the short name of the table, the fully qualified name, write attributes, and keys.

Anything that creates an object from scratch (not by replicating an existing object and not by changing the entire file - which is a File function) is an application-specific action. Anything that initiates an activity that is separate from filing, viewing, editing and properties can be represented in the control area either as separate buttons or as items on button menus.

In a pop-up window we don't allow the use of menus. We group related commands in menus. Before menus are generated we try to fit every action in a single button. If the width of the buttons is bigger than the width of the window then we group the buttons in the menus. There are three different possible items for the menus. An item, which issues a direct command is the same as a button, with the only difference that it is now an item in a menu; command item. A item with an little arrow pointing to the right has a sub-menu; menu item. A item with three dots at the end of the name is a window item. A window item pop-ups a command window, when it is selected. The menu advise generated in the control selection is used for the grouping of the actions in menus. To create a menu advise we use the pre- and post conditions, the parameter list of the action and large glossaries with frequently used names of actions in the standard menus. Pre- and post conditions capture some semantics of the action. Actions with similar pre- and post conditions should be grouped in a menu. The parameter list of the actions shows us which actions are using the same input or output parameters. Actions which cannot be placed in one of the standard menus are placed in an 'Application' menu. The menu organizer tries to group related actions in one of the Application menus. After the actions have been placed in the menus the user interface designer can easily adjust the generated menus.

Building a bridge between DevGuide and D2M2edit

DevGuide is the user interface development tool of Sun Microsystems(Figure 5). DevGuide has a palette from, which the user interface designer can drag objects onto the screen. For the specification of the attributes of each different interaction object DevGuide uses property sheets. D2M2-Edit is the Delft University of Technology's Direct Manipulation Manager Editor[Beek90], which is an interactive graphics editor implemented in Open Windows for creating semantic data diagrams (node and arc style figures). An application designer will use D2M2-Edit to create a semantic definition of an application: all objects, relations between objects (both part-whole relations and class hierarchy relations), actions on objects, pre- and postconditions on actions, and attributes (ie, properties) of objects. Every object has its own property-sheet in which the attributes, actions and conditions will be input as text and viewed as scrolling lists. The user interface designer will select one or more actions and attributes which are to be represented in a base window or pop-up window, and then drag the selection on top of a window or on the palette of DevGuide.(Figure 6). DevGuide will then invoke functions, that apply the style guide rules to choose appropriate controls, configure the chosen controls, and place the controls. If data is dropped on the palette, then a new pop-up window will be created with a control area. The goal of the layout of single-column pop-up windows will be to reinforce logical relationships (specified to D2M2-Edit as part of the application semantics) with visual cues, to create an aesthetically-pleasing layout.Once this is done, the user interface designer will be able to use DevGuide to modify or refine the design, and to integrate in coded application procedures. New data can be added an dragged onto already existing windows which already contain widgets.

Status of project

The first phase, to be completed in June 1991, will integrate D2M2-Edit and DevGuide (Figure 5). The rules dealt with in phase one will deal with:

- selection of controls for pop-up windows.
- placement of controls within pop-up windows, assuming a single-column layout for the window
- ordering of items within a menu, including grouping by related functionality

In phase one, the designer will choose what is to be assigned to a given base window or pop-up, and the DevGuide extensions will make design decisions. The designer will them be able to interact with DevGuide as at present, but DevGuide will not carry with it the semantics brought over from D2M2-Edit.

In phase two, DevGuide will carry semantic information so that, for instance, additional controls can be added to a pop-up and a new layout can then be created combining the old and the new controls. Also, in phase two, additional rules will be developed. This may also mean that additional information will be needed to be entered into D2M2-Edit. The additional rules will deal with issues such as:

- allocation of menu items to multiple pop-up menus, one per button menu (based on semantics of the menu items)
- hierarchical structuring of multiple levels of pop-up menus (based on semantics of the menu items)
- grouping of functional related controls in a pop-up window, by using white spacing.

Conclusions and Future work

Automatic generation of user interface from application data is a logical step and useful for the development of new user interfaces. Coupling application design and user interface design is a step to bring the traditional software engineering and user interface design together. The automatic generation tool can be integrated with other tools and toolkits. The presentation component (Figure 2) is currently integrated with DevGuide. This back end has to be adjusted for other tools. This project just started this year and there are some topics and issues, which we want to work on:

- Integration of automatic generation tool in a framework for the development of whole UIMS.A typical example is the UIDE[Fole91], in which one shared knowledge base is being used for the complete specification of the application and the user interface.
- The logical order in a property sheet is now determined by the designer. He tries to create a top-down visual cue. The order of controls in a pop-up window and in menus is often based on interdependency relations. These interdependencies are partly captured in the pre- and post conditions of the data elements. Pre- and post conditions are now being used for grouping related actions in the menus. The designer creates dialog box now by selecting controls for the designated dialog box.In the future he will select hundreds of data elements, which all have interdependencies. The automatic generation tool will then decide on the placement of controls in different dialog boxes.
- Currently we used the OPEN LOOK standard for the control selection and the design of the user interface. For special purpose applications special user interface needs may be required. Therefore, we are going to develop a tool for domain specific design.

Acknowledgments

First Dennis de Baar wants to thank Jim Foley for his support and help during his research and writing. We want to thank students at GeorgiaTech for their feedback: Daniel Gieskens, Wonchul Kim, Noi Sukaviriya, Srdjan Kovacevic, Mark Grey and Jens Kilian.

Partial funding for this project was provided by Sun Microsystems' Collaborative Research program, and by the National Science Foundation Fund Grant # IRI -8813179. We thank Bob Ellis of Sun for his skillful management of our research proposal, Kevin Mullet for his feedback and suggestions and Bob Watson for arranging software access. We also want to thank the Technical University of Delft for letting us use D2M2edit and especially Charles van der Mast and Johan Versendaal for their feedback.

References

[Appl86] Apple Computer, Inc. *Human Interface Guidelines: The apple Desktop Interface.* Apple Programmers and developer's Association. Renton, WA. 1986.

[Beek90] Beekman, W.H.R. *D2m2edit, Master's Thesis.* Delft University of Technology, July 1990

[Fole88] Foley, J.D., C. Gibbs, W. Kim, S. Kovacevic, L. Moran, P. Sukaviriya, *A Knowledge-Based User Interface Management System*, CHI'88 Proceedings, Washington DC, may 1988, pp. 67-72.

[Fole91] Foley, J.D., W. C. Kim, S. Kovacevic, and K. Murray, *UIDE - An intelligent User Interface Design Environment*, in Sullivan, J and S. Tyler (eds.), Architectures for Intelligent Interfaces: Elements and Prototypes, Addison-Wesley, 1991.

[Hewl90] Hewlett-Packard Company, *HP Interface Architect Developer's Guide*, Hewlett-Packard Company, Corvallis, Oregon, October 1990.

[IBM87] IBM Corporation. *System Application Architecture, Common Access Panel Design and User Interaction.* SC26-4351-0. December 1987.

[Olse89] Olsen, D. *A programming Language Basis for User Interface Management*,CHI'89 Proceedings, Austin, Texas, may 1989, pp. 171-176.

[OSF] Open Software Foundation. *OSF/Motif Style Guide*, Revision 1.0, OSF 11 Cambridge Center, Cambridge, MA 02142, ISBN 0-13-640491-X, 1990.

[Sun90a] Sun Microsystems, Inc. and AT&T OPEN LOOK, *Graphical User Interface Application Style Guidelines.* Addision-Wesley Publishing Company, Inc. ISBN 0-201-52364-7, 1990.

[Sun90b] Sun Microsystems, Inc., *Open Windows Developer's Guide 1.1, Reference Manual*, Part No. 800-5380-10, Revision A, of June 1990.

[Wiec89] Wiecha, C., W. Bennett, S. Boies, and J. Gould, *Generating Highly Interactive User Interfaces*, CHI'89 Proceedings, Austin, Texas, may 1989, pp. 277-282.

[Zand90] Zanden, B. Vander and B.A. Myers, Automatic, *Look-and-Feel Independent Dialog Creation for graphical User Interfaces*, CHI'90 Proceedings, Seattle, Washington, April 1990, pp. 27-34.